BASE DI DATI PER KALEN DARIO

Giulia Stazio, Davide Tonziello

Luglio 2025

Contents

1 Introduzione **2**

2 Specifica dei requisiti **2**

3 Progettazione concettuale **4**

4 Ristrutturazione **5**
4.1 Analisi delle ridondanze 5
4.2 Eliminazione di: generalizzazioni, attributi composti e multi valore 5
4.3 Partizione e accorpamento di entità e associazioni 5
4.4 Scelta degli identificatori principali 5
4.5 Diagramma ristrutturato 6
4.6 Dizionario classi 7
4.7 Dizionario relazioni 9

5 Progettazione logica (modello relazionale) **10**

6 Progettazione fisica **11**
6.1 Creazione delle tabelle 11
6.2 Constraints 12
6.3 Procedures 12
6.4 Trigger 14
6.5 PLSQL Functions 15
6.6 SQL Functions 17

1 Introduzione

Questo è lo sviluppo di una base di dati per un'applicazione chiamata 'Kalen Dario'. In questa applicazione un utente potrà gestire i propri impegni sotto forma di ToDo, organizzandoli in Bacheche apposite, con la possibilità di modificarli e condividerli con altri utenti.

2 Specifica dei requisiti

In seguito sarà riportata in corsivo la traccia fornita come assignment:

Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing) che consente di tenere traccia delle attività personali da svolgere (nel seguito chiamate semplicemente "ToDo"), ispirato al software Trello. Il software deve consentire all'utente di organizzare e gestire le proprie attività personali in modo efficiente, utilizzando un'interfaccia intuitiva e flessibile. L'utente entra nel sistema specificando la propria login e password, che devono essere univoche.
Si suppone che soltanto il nome utente debba essere univoco.

I ToDo sono organizzati in tre bacheche (ogni bacheca ha un titolo e una descrizione), che possono essere create, modificate ed eliminate. I titoli delle tre bacheche sono Università, Lavoro e Tempo Libero.
Si suppone che Università, Lavoro e Tempo libero siano tre formati di bacheche. In fase di creazione di una bacheca, se non specificato, il formato sarà impostato come Default(un quarto formato).

I ToDo all'interno di una bacheca sono ordinati, secondo un ordine modificabile dall'utente. L'utente può creare, modificare ed eliminare ToDo, così come può spostare un ToDo da una bacheca all'altra oppure cambiarne la posizione all'interno della bacheca. Ogni ToDo ha un titolo e una data di scadenza, un link ad una URL correlata all'attività, una descrizione dettagliata e un'immagine. Tutti questi elementi sono opzionali e possono essere modificati in qualsiasi momento.
Gli attributi saranno quindi tutti riutilizzabili all'interno di diversi ToDo, inoltre verrà reso possibile modificare l'ordine dei ToDo all'interno di una bacheca tramite funzioni di sort, quali Alfabetico(titolo), Data di scadenza o stato di completamento. Inoltre sarà possibile scegliere l'immagine tra un insieme di immagini predefinite e si terrà traccia all'interno della base di dati soltanto dell'indice di questa.

Ogni ToDo può contenere inoltre una lista di altri utenti che condividono quel ToDo. In pratica quel ToDo comparirà nella bacheca corrispondente di ognuno di tali utenti . Ad esempio, se Pippo, Pluto e Paperino condividono un ToDo, ed esso si trova nella bacheca Tempo Libero di Pippo, allora esso comparirà anche nelle bacheche Tempo Libero di Pluto e Paperino. Ogni utente può leggere chi sono gli altri utenti con i quali il ToDo è condiviso. L'autore del ToDo può aggiungere o eliminare condivisioni.

Visto che le bacheche possono essere create, eliminate e modificate, non è detto che utenti diversi abbiano tutte le bacheche in comune. Pertanto in fase di condivisione di un ToDo, questo verrà inserito nella bacheca di Dafault del ricevente e sarà quest'ultimo a poterlo spostare.

Infine il ToDo ha un colore di sfondo che viene mostrato nell'interfaccia grafica. L'utente deve poter scrivere e modificare ognuna di tali informazioni. Un ToDo può essere settato come completato oppure come non completato (di default è non completato).
Del colore di sfondo, proprio come per l'immagine, si terrà traccia solo dell'indice correlato al colore selezionato.

Il sistema deve poter fornire su richiesta dell'utente, l'elenco di ToDo in scadenza nella giornata odierna, oppure quelli in scadenza entro un certo giorno specificato dall'utente. Il sistema deve consentire anche la ricerca per nome o per titolo dei ToDo. Per i ToDo scaduti, il sistema mostra il nome del ToDo in rosso, per evidenziare il superamento della scadenza prevista.
Saranno implementate funzioni per elencare ToDo in scadenza in una giornata specifica. Per quanto riguarda i ToDo scaduti, il colore verrà impostato automaticamente a rosso, sovrascrivendo la scelta dell'utente(il rosso non sarà fra le opzioni di selezione dell'utente).

3 Progettazione concettuale

Il class diagram di partenza.

Un utente può creare un numero arbitrario di ToDo e condividerli con altri utenti tramite la relazione placement. Una bacheca può contenere un numero arbitrario di ToDo e i ToDo possono appartenere a più bacheche contemporaneamente, ma non meno di una.

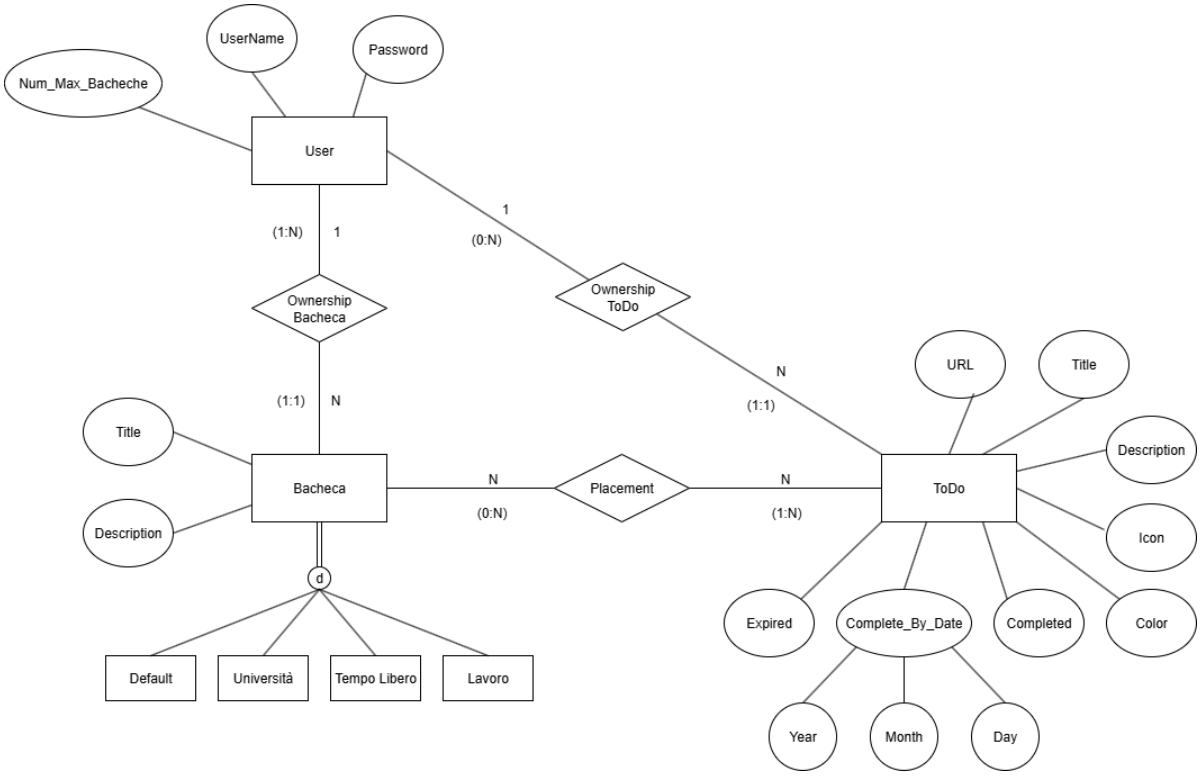


Figure 1: diagramma concettuale di partenza

4 Ristrutturazione

4.1 Analisi delle ridondanze

L'attributo *Expired* è una ridondanza in quanto è calcolabile tramite *Complete_By_Date* e la data corrente, ma è utile da tenere per evitare di dover fare controlli su date ogni volta che si accede al ToDo.
Non sono presenti altre ridondanze.

Da notare che la relazione *Ownership ToDo* non è una ridondanza in quanto in una Bacheca sono presenti i ToDo visualizzabili dal suo proprietario, anche se non ne è il creatore.

4.2 Eliminazione di: generalizzazioni, attributi composti e multi valore

La generalizzazione disgiunta totale in bacheca può essere gestita portando i figli nel padre ed inserendo un attributo *Format*. In fase di creazione di una bacheca, laddove non specificata, il formato sarà impostato come Default. Non sono presenti altre generalizzazioni.

L'attributo composto Complete_By_Date è rappresentabile con il tipo DATE di SQL. Non sono presenti altri attributi composti.

Non sono presenti attributi multi valore.

4.3 Partizione e accorpamento di entità e associazioni

Non è stato ritenuto necessario effettuare partizioni o accorpamenti.

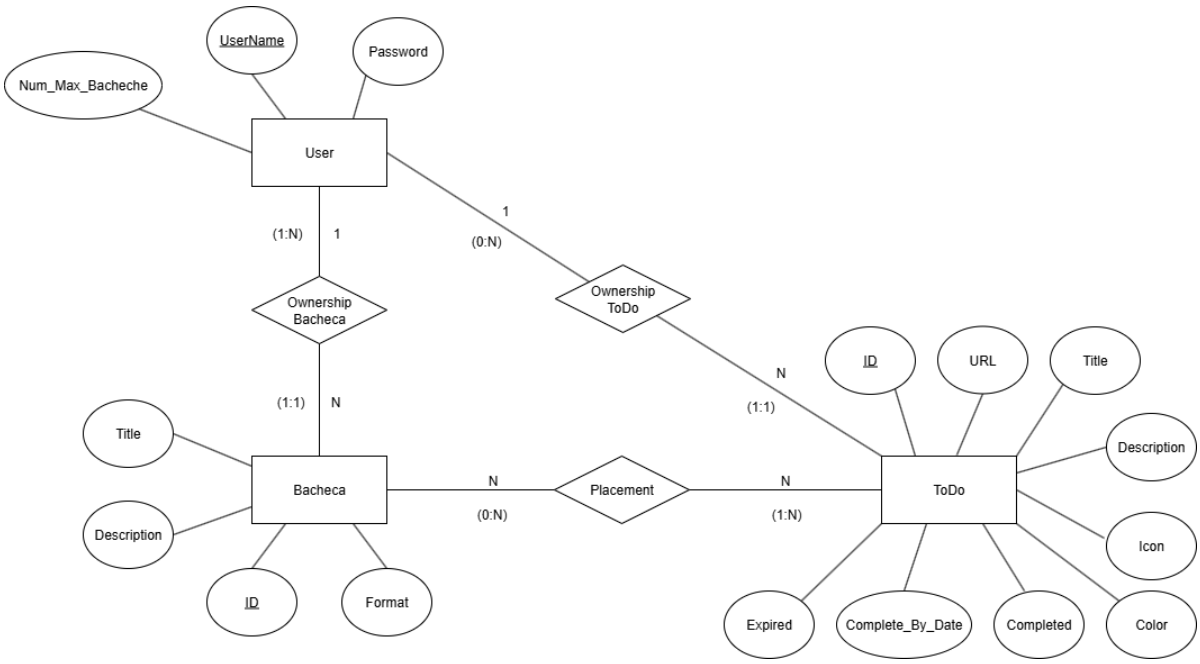
4.4 Scelta degli identificatori principali

Per l'entità User è stato scelto l'attributo Username.

Per le entità Bacheca e ToDo sono stati scelti degli ID in quanto tutti gli attributi presenti devono poter essere riutilizzati.

Ad esempio possono esistere Bacheche con lo stesso Title, Description, ecc. Lo stessa cosa vale per un ToDo.

4.5 Diagramma ristrutturato



4.6 Dizionario classi

Nome classe	Descrizione	Attributi
User	Un utente dell'applicazione	- Username(varchar(100)): Il nome scelto dall'utente - Password(varchar(100)): La password dell'account - Num_Max_Bacheche(int): Il numero massimo di bacheche che si possono avere contemporaneamente
Bacheca	Una bacheca di un utente	- ID(int): L'identificativo univoco di una bacheca - Title(varchar(100)): Il titolo della bacheca - Description(varchar(1000)): La descrizione della bacheca - Format(varchar(12)): Il formato della bacheca. A seconda del formato, le bacheche si comporteranno in modo diverso nell'interfaccia - IsDefault(boolean): Indica se la bacheca è quella predefinita

ToDo	I Todo creati dagli utenti	- ID(int): L'identificativo univoco di un ToDo - Title(varchar(100)): Il titolo del ToDo - URL(varchar(2048)): Un eventuale URL relativo al ToDo - Description(varchar(1000)): La descrizione del ToDo - Color(int): Un intero che indica quale colore è stato scelto come Background del ToDo (ai colori disponibili sono associati dei numeri da 0 a 4) - Icon(int): Un intero che indica quale icona è associata al ToDo. Le icone dei file png numerati che si trovano in una apposita cartella - Complete_By_Date(date): La data entro la quale deve (o doveva) essere stato completato il ToDo - Completed(boolean): Un flag che indica se il ToDo è stato completato - Expired(boolean): Un flag che indica se il ToDo è scaduto

4.7 Dizionario relazioni

Nome Relazione	Descrizione	Attributi
Placement	La presenza di un ToDo in una bacheca	- IDBacheca(int): L'identificativo della bacheca in cui si trova il ToDo referenziato da IDToDo - IDToDo(int): L'identificativo del ToDo contenuto nella bacheca referenziata da IDBacheca
Ownership Bacheca	La proprietà di una bacheca	- Username(varchar(100)): Il proprietario della bacheca - IDBacheca(int): La bacheca posseduta
Ownership ToDo	La proprietà di un ToDo	- Username(varchar(100)): Il proprietario del ToDo - IDToDo(int): Il ToDo posseduto

5 Progettazione logica (modello relazionale)

Le relazioni *Ownership Bacheca* e *Ownership ToDo* sono relazioni 1:N e verranno implementate con una chiave esterna(Owner) inserita rispettivamente nelle tabelle Bacheca e ToDo. La relazione *Placement* invece, essendo N:N, necessita di una tabella a sé stante.

ENTITÀ:

USER:(UserName, Password, Num_Max_Bacheche)

BACHECA:(ID, Description, Owner, Title, Format)
BACHECA.Owner → USER.UserName

TODO:(ID, URL, Title, Description, Owner, Icon, Color, Complete_By_Date)
TODO.Owner → USER.UserName

RELAZIONI:

PLACEMENT:(IDBacheca, IDToDo)
PLACEMENT.IDBacheca → BACHECA.ID
PLACEMENT.IDToDo → TODO.ID

6 Progettazione fisica

Di seguito è presente il codice per la creazione e gestione del DataBase, alcune considerazioni:

-Vengono utilizzate le " per termini considerati "reserved" in SQL

-Per la creazione di funzioni e trigger è utile inserire *OR REPLACE* dopo *CREATE* per facilitare l'apporto di eventuali modifiche in futuro, riutilizzando la stessa signature

6.1 Creazione delle tabelle

Tabella USER:

```
CREATE TABLE "user"
(
UserName varchar(100) PRIMARY KEY,
"Password" varchar(100) NOT NULL,
Num_Max_Bacheche INTEGER DEFAULT 10
)
```

Tabella BACHECA:

```
CREATE TABLE bacheca
(
"ID" int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
Title varchar(100),
Description varchar(1000),
"Owner" varchar(100) REFERENCES "user"(UserName) ON DELETE
    CASCADE,
Format varchar(12) DEFAULT 'Default',
IsDefault boolean DEFAULT false
)
```

Il cascade su "Owner" fa in modo che all'eliminazione di una tupla dalla tabella User, vengano eliminate anche tutte le tabelle associate a quell'utente

Tabella TODO:

```
CREATE TABLE todo
(
"ID" int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
Title varchar(100),
URL varchar(256),
Description varchar(1000),
"Owner" varchar(100) REFERENCES "user"(UserName) ON DELETE
    CASCADE,
Icon int DEFAULT 0,
Color int DEFAULT 0,
Complete_By_Date date,
Completed boolean DEFAULT false,
Expired boolean DEFAULT false
)
```

Il cascade su "Owner" qui serve per fare in modo che alla rimozione di un Utente dalla tabella User vengano eliminati anche tutti i ToDo creati da esso

Tabella PLACEMENT:

```
CREATE TABLE placement
(
IDBacheca int REFERENCES bacheca("ID") ON DELETE CASCADE,
IDToDo int REFERENCES todo("ID") ON DELETE CASCADE,
PRIMARY KEY (IDBacheca, IDToDo)
)
```

Il cascade su IDBacheca fa in modo che all'eliminazione di una bacheca, vengano rimosse anche le tuple della tabella placement che la referenziano. La stessa cosa vale per il cascade su IDToDO e l'eliminazione di un ToDo. Da notare che dopo l'eliminazione di una bacheca, non vengono rimossi automaticamente i ToDo associati dalla tabella ToDo. Di questa operazione, si occuperà un trigger apposito.

6.2 Constraints

I constraint sono stati implementati in fase di creazione delle tabelle con le operazioni CASCADE, vincoli di integrità (predefiniti) e trigger.

6.3 Procedures

La seguente procedura permette di modificare la bacheca di default in cui vengono inseriti i ToDo condivisi dagli altri utenti

```
CREATE OR REPLACE PROCEDURE changeDeafultBacheca
```

```
        (idnewdefault bacheca."ID"%TYPE, proprietario
            bacheca."Owner"%TYPE)
        as $$
        BEGIN
        UPDATE bacheca
        SET isdefault = false
        WHERE bacheca."Owner" LIKE proprietario AND isdefault = true;

        UPDATE bacheca
        set isdefault = true
        WHERE bacheca."ID" = idnewdefault;

        END;
        $$ language plpgsql;
```

Questa procedura si occupa di inserire un elemento nella tabella Placement in seguito ad una condivisione di un ToDo

```
        CREATE OR REPLACE PROCEDURE condividiTodo(utente varchar(100),
            idtodocondiviso int)
        as $$
        DECLARE
        destination int;
        BEGIN

        SELECT "ID" INTO destination
        FROM bacheca
        WHERE "Owner" LIKE utente AND isdefault = true;

        INSERT INTO placement (idbacheca, idtodo) values
        (destination, idtodocondiviso);

        END;
        $$ language plpgsql;
```

Questa procedura invece si occupa di rimuovere condivisioni

```
        CREATE OR REPLACE PROCEDURE annullaCondivisioneTodo(utente
            varchar(100), idtodocondiviso int)
        AS $$
        BEGIN
        DELETE FROM placement
        WHERE idbacheca IN (SELECT "ID" FROM bacheca WHERE "Owner" LIKE
            utente) AND idtodo = idtodocondiviso;
        END;
        $$ language plpgsql
```

La seguente procedura si occupa di spostare un ToDo da una tabella all'altra

```
CREATE OR REPLACE PROCEDURE moveTodo(todotomove int, origin
    int, destination int)
as $$
BEGIN
UPDATE placement
SET idbacheca = destination
WHERE idtodo = todotomove AND idbacheca = origin;
END;
$$ language plpgsql;
```

Questa è una procedura fittizia che serve unicamente ad attivare il trigger *checkExpiration* e verrà chiamata all'avvio e al refresh dell'applicazione. Viene "modificato" l'attributo completed in quanto è del tipo di dato più semplice (boolean) e quindi più leggero da maneggiare

```
CREATE OR REPLACE PROCEDURE checkAllExpiredDates() AS $$
DECLARE
tem record;
BEGIN
FOR tem IN (SELECT * FROM todo)
LOOP
UPDATE todo
SET completed = tem.completed
WHERE "ID"= tem."ID";
END LOOP;
END;
$$ language plpgsql;
```

6.4 Trigger

Il seguente trigger viene chiamato ogni volta che si effettua una rimozione dalla tabella Placement

```
CREATE OR REPLACE TRIGGER removeToDo AFTER DELETE ON placement
FOR EACH ROW
EXECUTE FUNCTION removeToDoFunction();
```

Questo trigger viene attivato all'inserimento di un nuovo utente nella tabella User

```
CREATE OR REPLACE TRIGGER createDefaultBacheche AFTER INSERT ON
    "user"
FOR EACH ROW
```

```
3       EXECUTE FUNCTION createDefaultBachecheFunction();
```

Questo trigger si attiva all'inserimento di un Todo o ad un suo aggiornamento

```
1       CREATE OR REPLACE TRIGGER checkExpiration BEFORE INSERT OR
            UPDATE ON todo
2       FOR EACH ROW
3       EXECUTE FUNCTION checkExpirationFunction();
```

6.5 PLSQL Functions

Questa funzione verifica la scadenza di un ToDo. Verrà chiamata da un apposito trigger all'inserimento/modifica di un ToDo

```
1       CREATE OR REPLACE FUNCTION checkExpirationFunction() RETURNS
            TRIGGER AS $$
2       BEGIN
3       IF new.complete_by_date < current_date AND new.completed =
            false THEN
4       new.expired := true;
5       ELSE
6       new.expired := false;
7       END IF;
8       RETURN NEW;
9       END;
10      $$ language plpgsql;
```

Questa funzione verifica, in seguito ad una rimozione di un elemento dalla tabella Placement, se l'utente ad aver rimosso il ToDo ne è il creatore. In caso affermativo, lo elimina anche dalla tabella dei ToDo

```
1       CREATE OR REPLACE FUNCTION removeToDoFunction ()
2       RETURNS trigger
3       as $$
4
5       DECLARE
6       temporaneo1 varchar(100);
7       temporaneo2 varchar(100);
8
9       BEGIN
10      Select "Owner" into temporaneo1
11      From bacheca
12      where bacheca."ID" = OLD.IDBacheca;
13
14      Select "Owner" into temporaneo2
15      From todo
```

```
        Where todo."ID" = OLD.IDToDo;

        IF temporaneo1 LIKE temporaneo2 THEN
        Delete from todo where todo."ID" = OLD.IDToDo;
        END IF;
        RETURN OLD;
        END;
        $$ language plpgsql;
```

Questa funzione inserisce le quattro bacheche di partenza di un utente

```
        CREATE OR REPLACE FUNCTION createDefaultBachecheFunction()
            RETURNS trigger
        as $$

        BEGIN
        INSERT INTO bacheca (title, description, "Owner", Format,
            isDefault)
        values
        ('Default', 'your default bacheca', new.username, 'Default',
            true),
        ('Università', 'your university bacheca', new.username,
            'Università', false),
        ('Lavoro', 'your job bacheca', new.username, 'Lavoro', false),
        ('Tempo libero', 'your free time bacheca', new.username, 'Tempo
            libero', false);
        RETURN NULL;
        END;
        $$ language plpgsql;
```

Questa funzione restituisce gli ID dei ToDo che contengono la stringa ricercata (non case sensitive)

```
        CREATE OR REPLACE FUNCTION searchtodo(utente varchar(100),
            testo varchar(100))
        RETURNS integer ARRAY
        AS $$

        DECLARE
        ret integer ARRAY;
        tem int;

        BEGIN
        FOR tem in
        SELECT todo."ID"
        FROM Todo JOIN placement ON Todo."ID" = placement.idtodo JOIN
            bacheca ON bacheca."ID" = placement.idbacheca
```

```
13          WHERE bacheca."Owner" LIKE utente AND lower(Todo.title) LIKE
                lower('%'||testo||'%')
14
15          LOOP
16          ret := array_append(ret, tem);
17          END LOOP;
18          RETURN ret;
19          END;
20          $$ language plpgsql;
```

6.6 SQL Functions

Questa funzione restituisce una bacheca ordinata in ordine alfabetico (ascendente)

```
1           CREATE OR REPLACE FUNCTION sortAlphabetical(bachecaToSort int)
                RETURNS TABLE("ID" int, title varchar(100), url
                varchar(256), description varchar(1000), "Owner"
                varchar(100), icon int, color int, complete_by_date date,
                completed boolean, expired boolean)
2           AS $$
3           SELECT todo."ID", todo.title, todo.url, todo.description,
                todo."Owner", todo.icon, todo.color, todo.complete_by_date,
                todo.completed, todo.expired
4           FROM placement JOIN todo ON placement.idtodo = todo."ID"
5           WHERE placement.idbacheca = bachecaToSort
6           ORDER BY todo.title ASC;
7           $$ language SQL;
```

Questa funzione restituisce una bacheca ordinata in ordine di data di scadenza (ascendente)

```
1           CREATE OR REPLACE FUNCTION sortByDate(bachecaToSort int)
                RETURNS TABLE("ID" int, title varchar(100), url
                varchar(256), description varchar(1000), "Owner"
                varchar(100), icon int, color int, complete_by_date date,
                completed boolean, expired boolean)
2           AS $$
3           SELECT todo."ID", todo.title, todo.url, todo.description,
                todo."Owner", todo.icon, todo.color, todo.complete_by_date,
                todo.completed, todo.expired
4           FROM placement JOIN todo ON placement.idtodo = todo."ID"
5           WHERE placement.idbacheca = bachecaToSort
6           ORDER BY todo.complete_by_date ASC;
7           $$ language SQL;
```

Questa funzione restituisce i ToDo che scadono entro la data indicata (*treshold*)

```
1       CREATE OR REPLACE FUNCTION completeBefore(utente varchar(100),
            treshold date) RETURNS TABLE("ID" int, title varchar(100),
            url varchar(256), description varchar(1000), "Owner"
            varchar(100), icon int, color int, complete_by_date date,
            completed boolean, expired boolean)
2       AS $$
3       SELECT todo."ID", todo.title, todo.url, todo.description,
            todo."Owner", todo.icon, todo.color, todo.complete_by_date,
            todo.completed, todo.expired
4       FROM placement JOIN todo ON placement.idtodo = todo."ID" JOIN
            bacheca ON bacheca."ID" = placement.idbacheca
5       WHERE bacheca."Owner" = utente AND todo.complete_By_Date <=
            treshold;
6       $$ language SQL;
```

Questa funzione seleziona i ToDo di un utente scaduti o non scaduti a seconda del valore status (*status = true → scaduti ; status = false → non scaduti*)

```
1       CREATE OR REPLACE FUNCTION selectExpired(utente varchar(100),
            status boolean) RETURNS TABLE("ID" int, title varchar(100),
            url varchar(256), description varchar(1000), "Owner"
            varchar(100), icon int, color int, complete_by_date date,
            completed boolean, expired boolean)
2       AS $$
3       SELECT todo."ID", todo.title, todo.url, todo.description,
            todo."Owner", todo.icon, todo.color, todo.complete_by_date,
            todo.completed, todo.expired
4       FROM placement JOIN todo ON placement.idtodo = todo."ID" JOIN
            bacheca ON placement.idbacheca = bacheca."ID"
5       WHERE bacheca."Owner" = utente AND todo.expired = status;
6       $$ language SQL;
```

Questa funzione seleziona i ToDo di un utente completati o non completati a seconda del valore status (*status = true → completati ; status = false → non completati*)

```
1       CREATE OR REPLACE FUNCTION selectCompleted(utente varchar(100),
            status boolean) RETURNS TABLE("ID" int, title varchar(100),
            url varchar(256), description varchar(1000), "Owner"
            varchar(100), icon int, color int, complete_by_date date,
            completed boolean, expired boolean)
2       AS $$
3       SELECT todo."ID", todo.title, todo.url, todo.description,
            todo."Owner", todo.icon, todo.color, todo.complete_by_date,
```

```
        todo.completed, todo.expired
    FROM placement JOIN todo ON placement.idtodo = todo."ID" JOIN
        bacheca ON placement.idbacheca = bacheca."ID"
    WHERE bacheca."Owner" = utente AND todo.completed = status;
    $$ language SQL;
```